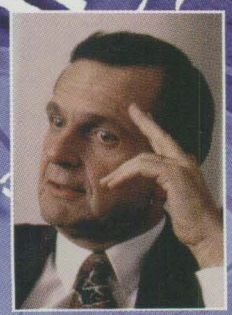
APRIL 1995 ENTERPRISE SYSTEMS JOURNAL VOLUME X • NUMBER 4

Focusing On IBM Host-Based Enterprise-Wide Computing

ENTERPRISE SYSTEMS JOURNAL

April 1995

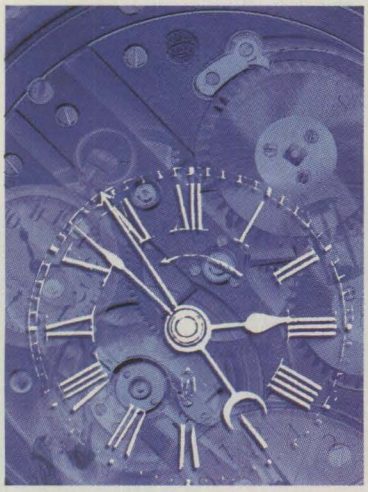


Dow Corning Manages Time Displacement In Data Center Consolidation

An Interview With Legent's Jerre Stead

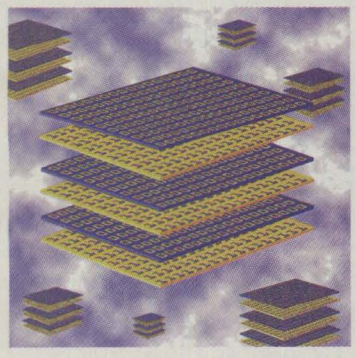
ESJ CONTENTS

April 1995 • Volume 10, Number 4



A 14-hour time difference can cause major problems when consolidating data centers. Find out how Dow Corning solved this problem on page 32. Cover illustration by Robert Kimmerle.

Test your client/server knowledge on page 14.



Client/server storage management on page 36.

I/S MANAGEMENT

OPERATING SYSTEM PLATFORMS

CLIENT/SERVER ENTERPRISE SOLUTIONS

DATABASE MANAGEMENT

STORAGE MANAGEMENT

FIRST IMPRESSION

DEPARTMENTS

REXX Coding Under OS/2

OS/2 contains a REXX interpreter. This article discusses how to code REXX under OS/2. The article starts with an overview of creating and running a REXX program under OS/2, followed by a brief discussion of some differences between REXX under MVS and REXX under OS/2. A discussion of DLL use in OS/2 follows. The article includes comments on OS/2 development tools and concludes with some comments on the future of REXX programming in the OS/2 environment.

Once a REXX compiler exists for OS/2, programmers can develop fast REXX applications that are easier to maintain and enhance than applications written in C.

By Lou Marco

Creating And Running REXX Programs Under OS/2

To create a REXX program under OS/2, you open an OS/2 editor (System or Enhanced) and enter REXX code. You must remember two things.

- Include a comment (/* ... */) as the first line in the REXX program.
- Name the program with an extension of CMD.

You may execute a REXX program in OS/2 by entering the name of the program (without the CMD extension) in an OS/2 command window or double clicking an icon associated with the program. To execute the REXX program from an OS/2 command window, place the program in a directory in the PATH assignment or execute it from its directory. Think of PATH assignments as the SYSEXEC, SYSPROC and ISPLLIB allocations rolled into one.

To view the directories in your PATH, type PATH in an OS/2 command window. To add a directory to the PATH, edit the data set CONFIG.SYS and append the directory containing the REXX program to the statement:

```
PATH=C:\OS2;...
```

If you want to change the CONFIG.SYS data set, make a backup copy *before* you change it.

To execute the REXX program from an icon, first decide where (in what folder) you want the icon to reside. Open that folder and open the Templates folder. Look for a Program icon in Templates and drag that icon to your folder. OS/2 will open the settings for the dragged icon. Enter the name of the REXX program in the Program notebook tab. Simply double click on this newly created icon and the REXX program runs.

REXX Differences Under MVS And OS/2

All REXX programming constructs (PARSE, IF/THEN/ELSE, DO/WHILE/UNTIL, SELECT/WHEN/OTHERWISE, CALL, etc.) behave the same under MVS and OS/2. Under both platforms, REXX uses the same built-in functions (POS, TRANSLATE, WORD, etc.). Both platforms support user-written functions and external REXX subroutine calls. REXX also passes host commands to the operating system in the same way. In short, the REXX language is identical under both platforms.

The major differences between MVS and OS/2 implementations deal with file I/O and the host operating system. REXX under OS/2 does not use EXECIO for file I/O. Instead, it uses four functions — LINEIN, LINEOUT, CHARIN and CHAROUT — for file I/O.

Use LINEIN and LINEOUT to read and write OS/2 files with carriage return/linefeed character line delimiters (text files). LINEIN and LINEOUT fetch a line at a time. Use CHARIN and

```
/** REXX EXEC to read a sequential, text file and write to screen **/
/** Using the CHARIN function.                                    **/
File_Name = "D:\REXXSTUF\REXXTEST.TXT"

Data = CHARIN( File_Name, 1, CHARS( File_Name ) )

SAY Data
LINEOUT( File_Name )                        /** This Closes the File **/
/** Using the LINEIN function                                     **/
A_Record = LINEIN( File_Name )
do while( LINES( File_Name \= 0 ))
     SAY A_Record
     A_Record = LINEIN( File_Name )
end /* do */
EXIT
```

Example 1: File Input With CHARIN And LINEIN

```
/* TESTWIN.CMD */
'@echo off'
call RxFuncAdd 'VInit', 'VREXX', 'VINIT'
initcode = VInit()
if initcode = 'ERROR' then signal Drop_Em

signal on failure name Drop_Em
signal on halt name Drop_Em
signal on syntax name Drop_Em

/* display the version number of VREXX */

ver = VGetVersion()
msg.0 = 1
msg.1 = 'VREXX version # ' ver
call VMsgBox 'TESTWIN.CMD', msg, 1

/* open a window and draw some text */

win.left   = 20
win.right  = 70
win.top    = 80
win.bottom = 40
id = VOpenWindow('My VREXX Window', 'RED', win)

text.1 = 'This is a VREXX window, created with a call to VOpenWindow.'
text.2 = 'The window currently has a title = My VREXX Window, and it'
text.3 = 'has a red background, which can be changed by a call to the'
text.4 = 'VBackColor.  The font is 12 point Times Roman.'

call VForeColor id, 'WHITE'
call VSetFont id, 'TIME', 12

x = 10
y = 900
do i = 1 to 4
   call VSay id, x, y, text.i
   y = y - 50
end

Drop_em:
    CALL Vexit
EXIT
```

Example 2: The VREXX DLL

CHAROUT to read and write files without these characters. CHARIN and CHAROUT read a file into or write a file from a REXX variable.

The REXX MVS programmer uses SYSDSN and LISTDSI to fetch information about MVS data sets. The REXX OS/2 programmer has similar capabilities. REXX under OS/2 contains functions that allow you to fetch information dynamically about OS/2 files and directories. Before you can use these functions, you must load them by calling the REXX OS/2 function ADDFUNCT. ADDFUNCT loads an OS/2 Dynamic Link Library (DLL) into memory. The next section discusses DLLs in greater detail.

See Example 1 for REXX code that uses the four I/O functions. Use SysFileTree to obtain the file size and read that number of characters with CHARIN. You must call RxFuncAdd to add SysFileTree function via a DLL. You do not need a DLL to Use CHARIN, LINEIN, CHAROUT or LINEOUT. Note the REXX program uses the CHARS function to determine the number of characters to read for character input and the LINES function to determine if any data exists for line input.

Using DLLs

OS/2 gives programmers two ways to use external function libraries. The first is the familiar method of linking external functions to form an executable program. You link object code — machine code produced by a compiler — with these external function libraries. The linker physically includes the external functions into the executable program at link time. The usual link process of generating an executable program is called a static link.

In contrast to libraries used in a static link, a DLL is a library that can be used by programs but is not part of these programs. Think of the DLL as programs made known to MVS applications by way of LIBPATH and ALTLIB commands. The functions in the DLL are not linked into the program. The functions in the DLL are dynamically loaded into memory on request at runtime. A program needing a function in a DLL issues a function call to load the DLL into memory. Once loaded, the programmer codes function calls to invoke the functions in the DLL.

Programmers invoke DLL functions by conforming to an Application Programming Interface (API). The API is a set of interfaces to the functions in the DLL. For an MVS example, think of ISPEXEC and ISPLINK as the API to Dialog Manager, or SQL as the API to DB2. Each DLL contains APIs that allow programmers writing in different programming languages to invoke the functions in the DLL. Thus, REXX programmers use one API and C programmers use another to invoke the same functions in the DLL. In short, knowing how to use the functions in a DLL boils down to knowing the API for that DLL for your particular

```
/** REXX Test Edit Macro */
"EXTRACT /getline"
"EXTRACT /Col"
Punct = "'/?.>,<;:+}+{\|=+-_)(*&+%$#@!" || '"'
This_Line = TRANSLATE( getline.1, " ", Punct )
Char_on_Cursor = SUBSTR( This_Line, Col.1, 1 )
IF Char_on_Cursor = " " THEN
   This_Word = WORD( SUBSTR( This_Line, Col.1 ), 1 )
ELSE
    DO
       Pos_of_First_Blank = LASTPOS( " ", SUBSTR( This_Line, 1, Col.1 )
       IF Pos_of_First_Blank = 0 THEN
          This_Word = WORD( This_Line, 1 )
       ELSE
          This_Word = WORD( SUBSTR( This_Line, Pos_of_First_Blank ), 1)
    END
"C:\OS2\VIEW.EXE REXX.INF " This_Word
```

Example 3: REXXHELP Edit Macro

programming language.

Since REXX under OS/2 is an interpreter, not a compiler, REXX programmers under OS/2 must use DLLs to use non-REXX external routines. REXX programmers do not need DLLs to call external REXX routines (provided these external REXX routines are on the PATH or in the current directory).

You should purge the DLL from memory following successful or unsuccessful execution. If not, you will have various DLLs loaded in memory, consuming precious resources. Issue a function call to purge the DLL from memory.

There are pros and cons to both static and dynamic link libraries. Until a REXX compiler exists for OS/2, the question is moot. However, a REXX compiler for OS/2 someday may be available, so a few words on these pros and cons may be useful.

One favorable feature of the static link is that your executable program loads and executes more quickly. The program using a DLL begins execution, then stops when OS/2 loads the DLL. If the DLL is already loaded, OS/2 has to search memory for the DLL. These are minor points unless performance is critical in your application.

Another favorable feature of the static link is that your program contains only those functions it needs. When you load a DLL, the entire DLL becomes memory-resident. This consumes precious resources. You cannot use just one function in the DLL without loading the entire DLL.

Another plus for static library use is the externally linked routines are already known to the linked program. The statically linked program need not code calls to load and purge the DLL — not to mention the code to do the requisite return code checking.

A favorable feature of DLLs is that they are global resources. That is, once loaded, any REXX program or executable program, running in any session, may use the functions in the DLL. OS/2 is smart enough to keep track of loaded DLLs and not load duplicate copies into memory. In contrast, statically linked programs may contain copies of the same routine. When these programs concurrently execute, the copies consume resources.

Another favorable feature of a DLL is that it normally (but not necessarily) contains APIs for several programming languages. Statically linked libraries are normally (but not necessarily) peculiar to a particular programming language.

Programmers using compilers may use static and dynamic libraries in their programs. Hopefully, REXX programmers can enjoy this flexibility in the near future.

REXX Development Tools For OS/2

Nearly all REXX development tools are a set of DLLs with documentation and on-line HELP. These tools are reasonably priced or, in some cases, free. This section discusses some of these tools. This list of REXX development tools is by no means exhaustive.

PMREXX

PMREXX is an application that allows you to run a REXX program in a window. PMREXX allows you to restart a REXX program, turn on tracing and save the results. A helpful PMREXX feature is that you can scroll the PMREXX window to see output that would be lost if the REXX program ran in an OS/2 window. PMREXX is distributed with OS/2 free of charge.

You can run PMREXX from an OS/2 command window. Enter PMREXX with a REXX program name. Remember, the REXX program must be in the current PATH assignment or in the current directory. You also can execute PMREXX from an icon. You can use the Program icon in Templates, the same as with an OS/2 REXX program. When you drag the Program template icon to your folder and OS/2 opens up the Settings notebook, enter PMREXX.EXE in the Program field and < > in the Parameters field. The angle brackets cause OS/2 to prompt for the name of the REXX program. You also may drop the REXX program icon on top of the PMREXX icon to invoke PMREXX for the REXX program.

VREXX

VREXX is a DLL that contains a REXX-only API to generate and display some PM objects. The VREXX user can display message boxes (with or without push buttons), radio buttons, check boxes, text entry fields and filename browsers. VREXX also contains a HELP file for on-line access to API documentation.

Fortunately, VREXX is public domain (send me a SASE and a 3.5-inch disk, and I will send it to you, with sample programs). Unfortunately, VREXX lacks the power needed for robust OS/2 PM applications, such as pull-down menus and scrollable window generation. VREXX is a good (and cheap) way to introduce the programmer to using API calls with REXX in OS/2.

Refer to Example 2 for a simple REXX program using VREXX. Most VREXX DLL functions require you to set a compound variable to some value. VREXX has about 50 routines that display PM objects.

Vis-PRO REXX

Vis-PRO REXX (Hockware, Cary, NC) is a complete OS/2 REXX application development package. Vis-PRO REXX supports all PM display objects.

Vis-PRO REXX programmers use a graphical metaphor to create applications. You create a Vis-PRO Form that holds all your application objects. Select the object from a palette, drag it to the

form and code the interface between the objects in REXX.

Vis-PRO REXX's strength is in creating new applications. It is not easy to use Vis-PRO REXX to retrofit existing REXX programs with PM display objects.

Of course, a tool as robust as Vis-PRO is not free.

VX-REXX

VX-REXX (Watcom, Waterloo, Ontario) is also a complete OS/2 REXX application development package. VX-REXX supports all PM display objects. VX-REXX generates executables that can run on any OS/2 system. In other words, you do not need DLLs to run a completed VX-REXX application.

It is easy to retrofit existing REXX programs that performed screen I/O to use PM objects with VX-REXX. All you do is replace the screen I/O calls with an API call to a PM display object.

VX-REXX is also not a freebie.

DA/2

IBM's Distributed Architecture/2 (DA/2) is a tool that allows REXX programmers to write Advanced Peer-to-Peer Communications (APPC), NetBIOS or named pipe applications. DA/2 supports combinations of single or multiple clients and servers (single client/single server, multiple clients/single server, etc.).

DA/2 comes with a DLL, on-line HELP and a Profile Editor. The Profile Editor allows the programmer to define characteristics of a client or server. These characteristics are not reflected in the API calls. This means REXX programmers developing an APPC application can write a client program, create a profile for the server as a named pipe and run the server program on their workstations. Developing the application now involves developing, running and debugging both the client and server programs on a single machine. Once both client and server programs run correctly, the programmer puts the server (or client) program on another machine and changes that Peer's characteristics in the Profile Editor. By changing a Peer's characteristics in the Profile Editor, REXX programmers can develop APPC applications between OS/2 and the mainframe.

DA/2 also contains an API for C programmers.

Miscellaneous Tools

OS/2 comes with a robust REXX online HELP subsystem. REXX HELP operates like OS/2 HELP. REXX HELP has examples for every function and REXX construct. The OS/2 Enhanced Editor helps REXX programmers by automatically entering REXX programming constructs. For example, when a programmer enters IF, the Enhanced Editor automatically generates THEN and ELSE.

REXX programmers can create macros for use in the Enhanced Editor. You can find some information on OS/2 REXX macro creation in the Quick Reference option of the Enhanced Editor Help menu. Refer to Example 3 for a macro that invokes REXX HELP for a desired REXX construct or function. To use this macro, place the cursor on a word while editing in the OS/2 Enhanced Editor. Pull down the COMMAND menu and select "Command Dialog" (or hit CNTL-I). The macro invokes OS/2 HELP on the selected word.

The Future Of REXX Under OS/2

Mainframe programmers who know REXX have a friend in REXX for OS/2. Their existing REXX knowledge, coupled with a good REXX application development tool, will get them up and running in OS/2 programming in short order.

Currently, C is the language of choice for many OS/2 developers. C programmers can get to any part of OS/2. Also, C programmers enjoy the benefits of using static and dynamic linked libraries. This speed and flexibility comes at a price; C applications, replete with pointers, can be difficult to maintain and enhance. Once a REXX compiler exists for OS/2, programmers can develop fast REXX applications that are easier to maintain and enhance than applications written in C.

Even in the absence of a REXX compiler, the old objections to developing applications in an interpretive language may disappear. The ever-increasing speeds of CPUs used in PCs may make the performance issue moot. Developers working on these fast machines may eschew C for REXX.

Today, REXX OS/2 programmers have several application development tools from which to choose. Tomorrow, REXX OS/2 programmers probably will have more from which to choose. The future looks bright for the REXX OS/2 programmer.

ABOUT THE AUTHOR

Lou Marco, a data processing training specialist for USAA, teaches DB, PL/I programming and ISPF Dialog Manager. He has 12 years experience and is the author of an upcoming book on REXX and Dialog Manager programming. USAA, 9800 Fredericksburg Rd., Bldg. A1 West, DP Training I, San Antonio, TX 78288, (210) 498-1372.